Package ‘covdepGE’

August 24, 2022

Title Covariate Dependent Graph Estimation

Version 1.0.0

Date 2022-08-14

Language en-US

BugReports `https://github.com/JacobHelwig/covdepGE/issues`

URL `https://github.com/JacobHelwig/covdepGE`

Description A covariate-dependent approach to Gaussian graphical modeling as described in Dasgupta et al. (2022). Employs a novel weighted pseudo-likelihood approach to model the conditional dependence structure of data as a continuous function of an extraneous covariate. The main function, covdepGE::covdepGE(), estimates a graphical representation of the conditional dependence structure via a block mean-field variational approximation, while several auxiliary functions (inclusionCurve(), matViz(), and plot.covdepGE()) are included for visualizing the resulting estimates.

License GPL (>= 3)

Encoding UTF-8

RoxygenNote 7.2.1

LinkingTo Rcpp, RcppArmadillo

Imports doParallel, foreach, ggplot2, glmnet, latex2exp, MASS, parallel, Rcpp, reshape2, stats

Suggests testthat (>= 3.0.0), covr, vdiffr

Config/testthat/edition 3

NeedsCompilation yes

Author Jacob Helwig [cre, aut],
Sutanoy Dasgupta [aut],
Peng Zhao [aut],
Bani Mallick [aut],
Debdeep Pati [aut]

Maintainer Jacob Helwig <`jacob.a.helwig@tamu.edu`>

Repository CRAN

Date/Publication 2022-08-24 08:22:38 UTC

R topics documented:

`covdepGE-package` *covdepGE: Covariate Dependent Graph Estimation*

Description

A covariate-dependent approach to Gaussian graphical modeling as described in Dasgupta et al. (2022). Employs a novel weighted pseudo-likelihood approach to model the conditional dependence structure of data as a continuous function of an extraneous covariate. The main function, covdepGE::covdepGE(), estimates a graphical representation of the conditional dependence structure via a block mean-field variational approximation, while several auxiliary functions (inclusionCurve(), matViz(), and plot.covdepGE()) are included for visualizing the resulting estimates.

Details

The conditional dependence structure (CDS) of a data matrix with p variables can be modeled as an undirected graph with p vertices, where two variables are connected if, and only if, the two variables are dependent given the remaining variables in the data. Gaussian graphical modeling (GGM) seeks to capture the CDS of the data under the assumption that the data are normally distributed. This distributional assumption is convenient for inference, as the CDS is given by the sparsity structure of the precision matrix, where the precision matrix is defined as the inverse covariance matrix of the data.

There is extensive GGM literature and many R packages for GGM, however, all make the restrictive assumption that the precision matrix is homogeneous throughout the data, or that there exists a partition of homogeneous subgroups. `covdepGE` avoids this strong assumption by utilizing information sharing to model the CDS as varying continuously with an extraneous covariate. Intuitively, this implies that observations having similar extraneous covariate values will have similar precision matrices.

To facilitate information sharing while managing complexity, `covdepGE` uses an efficient variational approximation conducted under the novel weighted pseudo-likelihood framework proposed by (1). `covdepGE` further accelerates inference by employing parallelism and executing expensive iterative computations in C++. Additionally, `covdepGE` offers a principled, data-driven approach for hyperparameter specification that only requires the user to input data and extraneous covariates to perform inference. Finally, `covdepGE` offers several wrappers around `ggplot2` for seamless visualization of resulting estimates, such as `matViz`, `inclusionCurve`, and the S3 method `plot.covdepGE`.

Author(s)

Maintainer: Jacob Helwig <jacob.a.helwig@tamu.edu>

Authors:

- Sutanoy Dasgupta <sutanoy@stat.tamu.edu>
- Peng Zhao <pzhao@stat.tamu.edu>
- Bani Mallick <bmallick@stat.tamu.edu>
- Debdeep Pati <debdeep@stat.tamu.edu>

References

(1) Sutanoy Dasgupta, Peng Zhao, Prasenjit Ghosh, Debdeep Pati, and Bani Mallick. An approximate Bayesian approach to covariate-dependent graphical modeling. pages 1–59, 2022.

See Also

Useful links:

- https://github.com/JacobHelwig/covdepGE
- Report bugs at https://github.com/JacobHelwig/covdepGE/issues

---

`covdepGE` *Covariate Dependent Graph Estimation*

Description

Model the conditional dependence structure of `X` as a function of `Z` as described in (1)

Usage

```
covdepGE(
  X,
  Z = NULL,
  hp_method = "hybrid",
  ssq = NULL,
  sbsq = NULL,
  pip = NULL,
  nssq = 5,
  nsbsq = 5,
  npip = 5,
  ssq_mult = 1.5,
  ssq_lower = 1e-05,
  snr_upper = 25,
  sbsq_lower = 1e-05,
  pip_lower = 1e-05,
  pip_upper = NULL,
  tau = NULL,
  norm = 2,
  center_X = TRUE,
  scale_Z = TRUE,
  alpha_tol = 1e-05,
  max_iter_grid = 10,
  max_iter = 100,
  edge_threshold = 0.5,
  sym_method = "mean",
  parallel = FALSE,
  num_workers = NULL,
  prog_bar = TRUE
)
```

Arguments

X	$n \times p$ numeric matrix; data matrix. For best results, n should be greater than p
Z	NULL OR $n \times q$ numeric matrix; extraneous covariates. If NULL, Z will be treated as constant for all observations, i.e.: `Z <- rep(0, nrow(X))` If Z is constant, the estimated graph will be homogeneous throughout the data. NULL by default
hp_method	`character in c("grid_search","model_average","hybrid")`; method for selecting hyperparameters from the the hyperparameter grid. The grid will be generated as the Cartesian product of ssq, sbsq, and pip. Fix X_j, the j-th column of X, as the response; then, the hyperparameters will be selected as follows: • If "grid_search", the point in the hyperparameter grid that maximizes the total ELBO summed across all n regressions will be selected • If "model_average", then all posterior quantities will be an average of the variational estimates resulting from the model fit for each point in the hyperparameter grid. The averaging weights for each of the n regressions are the exponentiated ELBO • If "hybrid", then models will be averaged over pip as in "model_average", with σ^2 and σ_β^2 chosen for each π in pip by maximizing the total ELBO over the grid defined by the Cartesian product of ssq and sbsq as in "grid_search" "hybrid" by default
ssq	NULL OR numeric vector with positive entries; candidate values of the hyperparameter σ^2 (prior residual variance). If NULL, ssq will be generated for each variable X_j fixed as the response as: `ssq <- seq(ssq_lower, ssq_upper, length.out = nssq)` NULL by default
sbsq	NULL OR numeric vector with positive entries; candidate values of the hyperparameter σ_β^2 (prior slab variance). If NULL, sbsq will be generated for each variable X_j fixed as the response as:

	`sbsq <- seq(sbsq_lower, sbsq_upper, length.out = nsbsq)` `NULL` by default
`pip`	`NULL` OR numeric vector with entries in $(0, 1)$; candidate values of the hyperparameter π (prior inclusion probability). If `NULL`, `pip` will be generated for each variable X_j fixed as the response as: `pip <- seq(pip_lower, pi_upper, length.out = npip)` `NULL` by default
`nssq`	positive integer; number of points to generate for `ssq` if `ssq` is `NULL`. `5` by default
`nsbsq`	positive integer; number of points to generate for `sbsq` if `sbsq` is `NULL`. `5` by default
`npip`	positive integer; number of points to generate for `pip` if `pip` is `NULL`. `5` by default
`ssq_mult`	positive numeric; if `ssq` is `NULL`, then for each variable X_j fixed as the response: `ssq_upper <- ssq_mult * stats::var(X_j)` Then, `ssq_upper` will be the greatest value in `ssq` for variable X_j. `1.5` by default
`ssq_lower`	positive numeric; if `ssq` is `NULL`, then `ssq_lower` will be the least value in `ssq`. `1e-5` by default

`snr_upper`	positive numeric; upper bound on the signal-to-noise ratio. If `sbsq` is `NULL`, then for each variable X_j fixed as the response:
	 `s2_sum <- sum(apply(X, 2, stats::var))`
	 `sbsq_upper <- snr_upper / (pip_upper * s2_sum)`
	 Then, `sbsq_upper` will be the greatest value in `sbsq`. `25` by default

`sbsq_lower`	positive numeric; if `sbsq` is `NULL`, then `sbsq_lower` will be the least value in `sbsq`. `1e-5` by default
`pip_lower`	numeric in $(0, 1)$; if `pip` is `NULL`, then `pip_lower` will be the least value in `pip`. `1e-5` by default

`pip_upper`	`NULL` OR numeric in $(0, 1)$; if `pip` is `NULL`, then `pip_upper` will be the greatest value in `pip`. If `sbsq` is `NULL`, `pip_upper` will be used to calculate `sbsq_upper`. If `NULL`, `pip_upper` will be calculated for each variable X_j fixed as the response as:
	 `lasso <- glmnet::cv.glmnet(X, X_j)`
	 `non0 <- sum(glmnet::coef.glmnet(lasso, s = "lambda.1se")[-1] != 0)`
	 `non0 <- min(max(non0, 1), p - 1)`
	 `pip_upper <- non0 / p`
	 `NULL` by default

`tau`	`NULL` OR positive numeric OR numeric vector of length n with positive entries; bandwidth parameter. Greater values allow for more information to be shared between observations. Allows for global or observation-specific specification. If `NULL`, use 2-step KDE methodology as described in (2) to calculate observation-specific bandwidths. `NULL` by default

`norm`	numeric in $[1, \infty]$; norm to use when calculating weights. `Inf` results in infinity norm. `2` by default
`center_X`	logical; if `TRUE`, center `X` column-wise to mean 0. `TRUE` by default
`scale_Z`	logical; if `TRUE`, center and scale `Z` column-wise to mean 0, standard deviation 1 prior to calculating the weights. `TRUE` by default
`alpha_tol`	positive numeric; end CAVI when the Frobenius norm of the change in the alpha matrix is within `alpha_tol`. `1e-5` by default
`max_iter_grid`	positive integer; if tolerance criteria has not been met by `max_iter_grid` iterations during grid search, end CAVI. After grid search has completed, CAVI is performed with the final hyperparameters selected by grid search for at most `max_iter` iterations. Does not apply to `hp_method = "model_average"`. `10` by default
`max_iter`	positive integer; if tolerance criteria has not been met by `max_iter` iterations, end CAVI. `100` by default
`edge_threshold`	numeric in $(0, 1)$; a graph for each observation will be constructed by including an edge between variable i and variable j if, and only if, the (i, j) entry of the symmetrized posterior inclusion probability matrix corresponding to the observation is greater than `edge_threshold`. `0.5` by default
`sym_method`	`character` in `c("mean","max","min")`; to symmetrize the posterior inclusion probability matrix for each observation, the (i, j) and (j, i) entries will be post-processed as `sym_method` applied to the (i, j) and (j, i) entries. `"mean"` by default
`parallel`	logical; if `TRUE`, hyperparameter selection and CAVI for each of the p variables will be performed in parallel using `foreach`. Parallel backend may be registered prior to making a call to `covdepGE`. If no active parallel backend can be detected, then parallel backend will be automatically registered using: `doParallel::registerDoParallel(num_workers)` `FALSE` by default
`num_workers`	`NULL` OR positive integer less than or equal to `parallel::detectCores()`; argument to `doParallel::registerDoParallel` if `parallel = TRUE` and no parallel backend is detected. If `NULL`, then: `num_workers <- floor(parallel::detectCores() / 2)` `NULL` by default
`prog_bar`	logical; if `TRUE`, then a progress bar will be displayed denoting the number of remaining variables to fix as the response and perform CAVI. If `parallel`, no progress bar will be displayed. `TRUE` by default

Value

Returns object of class `covdepGE` with the following values:

`graphs`	list with the following values:

- `graphs`: list of n numeric matrices of dimension $p \times p$; the l-th matrix is the adjacency matrix for the l-th observation
- `unique_graphs`: list; the l-th element is a list containing the l-th unique graph and the indices of the observation(s) corresponding to this graph
- `inclusion_probs_sym`: list of n numeric matrices of dimension $p \times p$; the l-th matrix is the symmetrized posterior inclusion probability matrix for the l-th observation
- `inclusion_probs_asym`: list of n numeric matrices of dimension $p \times p$; the l-th matrix is the posterior inclusion probability matrix for the l-th observation prior to symmetrization

`variational_params`

list with the following values:

- `alpha`: list of p numeric matrices of dimension $n \times (p-1)$; the (i, j) entry of the k-th matrix is the variational approximation to the posterior inclusion probability of the j-th variable in a weighted regression with variable k fixed as the response, where the weights are taken with respect to observation i
- `mu`: list of p numeric matrices of dimension $n \times (p-1)$; the (i, j) entry of the k-th matrix is the variational approximation to the posterior slab mean for the j-th variable in a weighted regression with variable k fixed as the response, where the weights are taken with respect to observation i
- `ssq_var`: list of p numeric matrices of dimension $n \times (p-1)$; the (i, j) entry of the k-th matrix is the variational approximation to the posterior slab variance for the j-th variable in a weighted regression with variable k fixed as the response, where the weights are taken with respect to observation i

`hyperparameters`

list of p lists; the j-th list has the following values for variable j fixed as the response:

- `grid`: matrix of candidate hyperparameter values, corresponding ELBO, and iterations to converge
- `final`: the final hyperparameters chosen by grid search and the ELBO and iterations to converge for these hyperparameters

`model_details` list with the following values:

- `elapsed`: amount of time to fit the model
- `n`: number of observations
- `p`: number of variables
- `ELBO`: ELBO summed across all observations and variables. If `hp_method` is `"model_average"` or `"hybrid"`, this ELBO is averaged across the hyperparameter grid using the model averaging weights for each variable
- `num_unique`: number of unique graphs
- `grid_size`: number of points in the hyperparameter grid
- `args`: list containing all passed arguments of length 1

`weights` list with the following values:

- `weights`: $n \times n$ numeric matrix. The (i, j) entry is the similarity weight of the i-th observation with respect to the j-th observation using the j-th observation's bandwidth
- `bandwidths`: numeric vector of length n. The i-th entry is the bandwidth for the i-th observation

Overview

Suppose that `X` is a p-dimensional data matrix with n observations and that `Z` is a q-dimensional extraneous covariate, also with n observations, where the l-th observation in `Z` is associated with the l-th observation in `X`. Further suppose that the l-th row of `X` follows a p-dimensional Gaussian distribution with mean 0 and precision matrix $\Omega(z_l)$, where z_l is the l-th entry of `Z` and Ω is a continuous function mapping from the space of extraneous covariates to the space of $p \times p$ non-singular matrices. Then, for the l-th observation, the (j, k) entry of $\Omega(z_l)$ is non-zero if, and only if, variable j and variable k are dependent given the remaining variables in `X`.

Given data satisfying these assumptions, the `covdepGE` function employs the algorithm described in (1) to estimate a graphical representation of the structure of Ω for each of the observations in `X` as a continuous function of `Z`. This graph contains an undirected edge between two variables X_j and X_k if, and only if, X_j and X_k are conditionally dependent given the remaining variables. Core components of this methodology are the weighted pseudo-likelihood framework in which inference is conducted via a block mean-field variational approximation.

Graph Estimation

Graphs are constructed using a pseudo-likelihood approach by fixing each of the columns X_j of `X` as the response and performing a spike-and-slab regression using the remaining variables X_k in `X` as predictors. To determine if an edge should be added between X_j and X_k, the posterior inclusion probability of X_k in a regression with X_j fixed as the response ($PIP_j(X_k)$) and vice versa ($PIP_k(X_j)$) are symmetrized according to `sym_method` (e.g., by taking the mean of $PIP_k(X_j)$ and $PIP_j(X_k)$). If the symmetrized PIP is greater than `edge_threshold`, an edge will be included between X_j and X_k.

To model Ω as a function of `Z`, n weighted spike-and-slab regressions are performed for each variable X_j fixed as the response. The similarity weights for the l-th regression are taken with respect to observation l such that observations having similar values of `Z` to z_l will have larger weights. These similarity weights in conjunction with the pseudo-likelihood framework comprise the weighted pseudo-likelihood approach introduced by (1). Note that model performance is best when $n > p$.

Variational Inference

Spike-and-slab posterior quantities are estimated using a block mean-field variational approximation. Coordinate Ascent Variational Inference (CAVI) is performed for each of the weighted regressions to select the variational parameters that maximize the ELBO. The parameters for each of the regression coefficients are the mean and variance of the slab (μ and σ^2_{var}, respectively) and the probability that the coefficient is non-zero (α). μ and α for all coefficients are initialized as 0 and 0.2, respectively.

CAVI for the n regressions is performed simultaneously for variable X_j fixed as the response. With each of the n sets of α as the rows of an $n \times (p - 1)$ matrix, the CAVI for variable X_j is ended for all n regressions when the Frobenius norm of the change in the α matrix is less than `alpha_tol` or after `max_iter` iterations of CAVI have been performed.

Note that since the regressions performed for variable X_j and X_k fixed as the response are independent of each other, they may be performed in parallel by setting `parallel = TRUE`. Registering parallel backend with greater than p workers offers no benefit, since each worker takes on one variable to fix as the response and perform the n regressions.

Hyperparameter specification

Each regression requires the specification of 3 hyperparameters: π (the prior probability of inclusion), σ^2 (the prior residual variance), and σ^2_β (the prior variance of the slab). `covdepGE` offers 3 methods for hyperparameter specification via the `hp_method` argument: `grid_search`, `model_average`, and `hybrid`. Empirically, `grid search` offers the best sensitivity and `model_average` offers the best specificity, while `hybrid` sits between the other two methods in both metrics.

The hyperparameter candidate grid is generated by taking the Cartesian product between `ssq`, `sbsq`, and `pip` (candidate values for σ^2, σ^2_β, and π, respectively). Each of the methods gives an approach for selecting points from this grid.

In `grid_search`, the point from the grid that produces the model that has the greatest total ELBO is selected, where the total ELBO is calculated by summing the ELBO for each of the n regressions for a variable X_j fixed as the response. Thus, all observations use the same set of hyperparameters for the regression on X_j.

Instead of selecting only one model as in `grid_search`, models are averaged over in `model_average`. With X_j fixed as the response, the unnormalized weights for each grid point used to perform this averaging is calculated by exponentiating the ELBO for each of the n regressions. Note that since the ELBO for a given grid point will vary across the n regressions due to differing similarity weights, each of the n sets of averaging weights will be unique.

Finally, `hybrid` combines `grid_search` and `model_average`. Fixing X_j as the response, for each π candidate in `pip`, the point in the grid defined by the Cartesian product of `ssq` and `sbsq` is selected by maximizing the total ELBO summed across the n regressions. The resulting models for each of the π candidates are then averaged using the exponentiated ELBO for each of the n regressions as the unnormalized averaging weights.

Note that in the search step of `grid_search` and `hybrid`, CAVI for each of the grid points is performed for at most `max_iter_grid` iterations. A second CAVI is then performed for `max_iter` iterations using the hyperparameters that maximized the total ELBO in the first step. Setting `max_iter_grid` to be less than `max_iter` (as is the default) will result in a more efficient search.

Candidate grid generation

The candidate grids (`ssq`, `sbsq`, and `pip`) may be passed as arguments, however, by default, these grids are generated automatically. Each of the grids are spaced uniformly between an upper end point and a lower end point. The number of points in each grid is 5 by default. Grids include end points, and the number of points in each grid is controlled by the arguments `nssq`, `nsbsq`, and `npip`. The lower endpoints (`ssq_lower`, `sbsq_lower`, and `pip_lower`) are all `1e-5` by default. The upper endpoints are calculated dependent on the variable X_j fixed as the response.

`ssq_upper` is simply the variance of X_j times `ssq_mult`. By default, `ssq_mult` is `1.5`.

`pip_upper` is calculated by regressing the remaining variables on X_j using LASSO. The shrinkage hyperparameter for LASSO is chosen to be `lambda.1se`. The number of non-zero coefficients estimated by LASSO is then divided by `p - 1` to calculate `pip_upper`. Note that if the LASSO estimate to the number of non-zero coefficients is 0 or $p - 1$, this estimate is changed to 1 or $p - 2$ (respectively) to ensure that `pip_upper` is greater than 0 and less than 1.

Finally, an upper bound is induced on σ_β^2 by deriving a rough upper bound for the signal-to-noise ratio that depends on σ_β^2. Let Σs_j^2 be the sum of the sample variances of the columns of the predictors X'. Under the simplifying assumptions that the expected values of X' and the spike-and-slab regression coefficients β are 0 and that X' and β are independent, the variance of the dot product of X' with β is $\pi \cdot \sigma^2 \cdot \sigma_\beta^2 \cdot \Sigma s_j^2$. Thus, the signal-to-noise ratio under these assumptions is given by $\pi \cdot \sigma_\beta^2 \cdot \Sigma s_j^2$. Replacing π with `pip_upper` and σ_β^2 with `sbsq_upper` gives an upper bound on the signal-to-noise ratio. Setting this bound equal to `snr_upper` gives an expression for `sbsq_upper`.

Similarity Weights

The similarity weight for observation k with respect to observation l is $\phi_{\tau_l}(||z_l - z_k||)$. Here, $||\cdot||$ denotes the norm specified by the `norm` argument, z_l and z_k are the values of `Z` for the l-th and k-th observations, ϕ_{τ_l} is the univariate Gaussian density with standard deviation τ_l, and τ_l is the bandwidth for the l-th observation.

`tau` may be passed as an argument, however, by default, it is estimated using the methodology given in (2). (2) describes a two-step approach for density estimation, where in the first step, an initial estimate is calculated using Silverman's rule of thumb for initializing bandwidth values, and in the second step, the density is refined by updating the bandwidth values. This methodology is used here to estimate the density of `Z`, and the updated bandwidths from the second step are used for `tau`.

References

(1) Sutanoy Dasgupta, Peng Zhao, Prasenjit Ghosh, Debdeep Pati, and Bani Mallick. An approximate Bayesian approach to covariate-dependent graphical modeling. pages 1–59, 2022.

(2) Sutanoy Dasgupta, Debdeep Pati, and Anuj Srivastava. A Two-Step Geometric Framework For Density Modeling. *Statistica Sinica*, 30(4):2155–2177, 2020.

Examples

```
library(ggplot2)

# get the data
set.seed(12)
data <- generateData()
X <- data$X
Z <- data$Z
interval <- data$interval
prec <- data$true_precision

# get overall and within interval sample sizes
n <- nrow(X)
n1 <- sum(interval == 1)
n2 <- sum(interval == 2)
n3 <- sum(interval == 3)

# visualize the distribution of the extraneous covariate
```

```
ggplot(data.frame(Z = Z, interval = as.factor(interval))) +
  geom_histogram(aes(Z, fill = interval), color = "black", bins = n %/% 5)

# visualize the true precision matrices in each of the intervals

# interval 1
matViz(prec[[1]], incl_val = TRUE) +
  ggtitle(paste0("True precision matrix, interval 1, observations 1,...,", n1))

# interval 2 (varies continuously with Z)
cat("\nInterval 2, observations ", n1 + 1, ",...,", n1 + n2, sep = "")
int2_mats <- prec[interval == 2]
int2_inds <- c(5, n2 %/% 2, n2 - 5)
lapply(int2_inds, function(j) matViz(int2_mats[[j]], incl_val = TRUE) +
         ggtitle(paste("True precision matrix, interval 2, observation", j + n1)))

# interval 3
matViz(prec[[length(prec)]], incl_val = TRUE) +
  ggtitle(paste0("True precision matrix, interval 3, observations ",
                 n1 + n2 + 1, ",...,", n1 + n2 + n3))

# fit the model and visualize the estimated graphs
(out <- covdepGE(X, Z, nssq = 2, nsbsq = 2, npip = 2))
plot(out)

# visualize the posterior inclusion probabilities for variables (1, 3) and (1, 2)
inclusionCurve(out, 1, 2)
inclusionCurve(out, 1, 3)
```

`generateData` *Generate Covariate-Dependent Data*

Description

Generate a 1-dimensional extraneous covariate and p-dimensional Gaussian data with a precision matrix that varies as a continuous function of the extraneous covariate. This data is distributed similar to that used in the simulation study from (1)

Usage

```
generateData(p = 5, n1 = 60, n2 = 60, n3 = 60, Z = NULL, true_precision = NULL)
```

Arguments

`p`	positive integer; number of variables in the data matrix. `5` by default
`n1`	positive integer; number of observations in the first interval. `60` by default
`n2`	positive integer; number of observations in the second interval. `60` by default
`n3`	positive integer; number of observations in the third interval. `60` by default

`Z`	`NULL` or numeric vector; extraneous covariate values for each observation. If `NULL`, `Z` will be generated from a uniform distribution on each of the intervals
`true_precision`	`NULL` OR list of matrices of dimension $p \times p$; true precision matrix for each observation. If `NULL`, the true precision matrices will be generated dependent on `Z`. `NULL` by default

Value

Returns list with the following values:

`X`	a `(n1 + n2 + n3)` $\times p$ numeric matrix, where the i-th row is drawn from a p-dimensional Gaussian with mean 0 and precision matrix `true_precision[[i]]`
`Z`	a `(n1 + n2 + n3)` $\times 1$ numeric matrix, where the i-th entry is the extraneous covariate z_i for observation i
`true_precision`	list of `n1 + n2 + n3` matrices of dimension $p \times p$; the i-th matrix is the precision matrix for the i-th observation
`interval`	vector of length `n1 + n2 + n3`; interval assignments for each of the observations, where the i-th entry is the interval assignment for the i-th observation

Extraneous Covariate

If `Z = NULL`, then the generation of `Z` is as follows:

The first `n1` observations have z_i from from a uniform distribution on the interval $(-3, -1)$ (the first interval).

Observations `n1 + 1` to `n1 + n2` have z_i from from a uniform distribution on the interval $(-1, 1)$ (the second interval).

Observations `n1 + n2 + 1` to `n1 + n2 + n3` have z_i from a uniform distribution on the interval $(1, 3)$ (the third interval).

Precision Matrices

If `true_precision = NULL`, then the generation of the true precision matrices is as follows:

All precision matrices have 2 on the diagonal and 1 in the $(2, 3)/(3, 2)$ positions.

Observations in the first interval have a 1 in the $(1, 2)/(1, 2)$ positions, while observations in the third interval have a 1 in the $(1, 3)/(3, 1)$ positions.

Observations in the second interval have 2 entries that vary as a linear function of their extraneous covariate. Let $\beta = 1/2$. Then, the $(1, 2)/(2, 1)$ positions for the i-th observation in the second interval are $\beta \cdot (1 - z_i)$, while the $(1, 3)/(3, 1)$ entries are $\beta \cdot (1 + z_i)$.

Thus, as z_i approaches -1 from the right, the associated precision matrix becomes more similar to the matrix for observations in the first interval. Similarly, as z_i approaches 1 from the left, the matrix becomes more similar to the matrix for observations in the third interval.

Examples

```
library(ggplot2)

# get the data
set.seed(12)
data <- generateData()
X <- data$X
Z <- data$Z
interval <- data$interval
prec <- data$true_precision

# get overall and within interval sample sizes
n <- nrow(X)
n1 <- sum(interval == 1)
n2 <- sum(interval == 2)
n3 <- sum(interval == 3)

# visualize the distribution of the extraneous covariate
ggplot(data.frame(Z = Z, interval = as.factor(interval))) +
  geom_histogram(aes(Z, fill = interval), color = "black", bins = n %/% 5)

# visualize the true precision matrices in each of the intervals

# interval 1
matViz(prec[[1]], incl_val = TRUE) +
  ggtitle(paste0("True precision matrix, interval 1, observations 1,...,", n1))

# interval 2 (varies continuously with Z)
cat("\nInterval 2, observations ", n1 + 1, ",...,", n1 + n2, sep = "")
int2_mats <- prec[interval == 2]
int2_inds <- c(5, n2 %/% 2, n2 - 5)
lapply(int2_inds, function(j) matViz(int2_mats[[j]], incl_val = TRUE) +
         ggtitle(paste("True precision matrix, interval 2, observation", j + n1)))

# interval 3
matViz(prec[[length(prec)]], incl_val = TRUE) +
  ggtitle(paste0("True precision matrix, interval 3, observations ",
                 n1 + n2 + 1, ",...,", n1 + n2 + n3))

# fit the model and visualize the estimated graphs
(out <- covdepGE(X, Z, nssq = 2, nsbsq = 2, npip = 2))
plot(out)

# visualize the posterior inclusion probabilities for variables (1, 3) and (1, 2)
inclusionCurve(out, 1, 2)
inclusionCurve(out, 1, 3)
```

`inclusionCurve` *Plot PIP as a Function of Index*

Description

Plot the posterior inclusion probability of an edge between two variables as a function of observation index

Usage

```
inclusionCurve(
  out,
  col_idx1,
  col_idx2,
  line_type = "solid",
  line_size = 0.5,
  line_color = "black",
  point_shape = 21,
  point_size = 1.5,
  point_color = "#500000",
  point_fill = "white"
)
```

Arguments

`out`	object of class `covdepGE`; return of `covdepGE` function
`col_idx1`	integer in $[1, p]$; column index of the first variable
`col_idx2`	integer in $[1, p]$; column index of the second variable
`line_type`	linetype; `ggplot2` line type to interpolate the probabilities. `"solid"` by default
`line_size`	positive numeric; thickness of the interpolating line. `0.5` by default
`line_color`	color; color of interpolating line. `"black"` by default
`point_shape`	shape; shape of the points denoting observation-specific inclusion probabilities; `21` by default
`point_size`	positive numeric; size of probability points. `1.5` by default
`point_color`	color; color of probability points. `"#500000"` by default
`point_fill`	color; fill of probability points. Only applies to select shapes. `"white"` by default

Value

Returns `ggplot2` visualization of inclusion probability curve

Examples

```
library(ggplot2)

# get the data
set.seed(12)
data <- generateData()
X <- data$X
Z <- data$Z
```

```
interval <- data$interval
prec <- data$true_precision

# get overall and within interval sample sizes
n <- nrow(X)
n1 <- sum(interval == 1)
n2 <- sum(interval == 2)
n3 <- sum(interval == 3)

# visualize the distribution of the extraneous covariate
ggplot(data.frame(Z = Z, interval = as.factor(interval))) +
  geom_histogram(aes(Z, fill = interval), color = "black", bins = n %/% 5)

# visualize the true precision matrices in each of the intervals

# interval 1
matViz(prec[[1]], incl_val = TRUE) +
  ggtitle(paste0("True precision matrix, interval 1, observations 1,...,", n1))

# interval 2 (varies continuously with Z)
cat("\nInterval 2, observations ", n1 + 1, ",...,", n1 + n2, sep = "")
int2_mats <- prec[interval == 2]
int2_inds <- c(5, n2 %/% 2, n2 - 5)
lapply(int2_inds, function(j) matViz(int2_mats[[j]], incl_val = TRUE) +
         ggtitle(paste("True precision matrix, interval 2, observation", j + n1)))

# interval 3
matViz(prec[[length(prec)]], incl_val = TRUE) +
  ggtitle(paste0("True precision matrix, interval 3, observations ",
                 n1 + n2 + 1, ",...,", n1 + n2 + n3))

# fit the model and visualize the estimated graphs
(out <- covdepGE(X, Z, nssq = 2, nsbsq = 2, npip = 2))
plot(out)

# visualize the posterior inclusion probabilities for variables (1, 3) and (1, 2)
inclusionCurve(out, 1, 2)
inclusionCurve(out, 1, 3)
```

matViz *Visualize a matrix*

Description

Create a visualization of a matrix

Usage

```
matViz(
  x,
  color1 = "white",
  color2 = "#500000",
  grid_color = "black",
  incl_val = FALSE,
  prec = 2,
  font_size = 3,
  font_color1 = "black",
  font_color2 = "white",
  font_thres = mean(x)
)
```

Arguments

`x`	matrix; matrix to be visualized
`color1`	color; color for low entries. `"white"` by default
`color2`	color; color for high entries. `"#500000"` by default
`grid_color`	color; color of grid lines. `"black"` by default
`incl_val`	logical; if `TRUE`, the value for each entry will be displayed. `FALSE` by default
`prec`	positive integer; number of decimal places to round entries to if `incl_val` is `TRUE`. `2` by default
`font_size`	positive numeric; size of font if `incl_val` is `TRUE`. `3` by default
`font_color1`	color; color of font for low entries if `incl_val` is `TRUE`. `"black"` by default
`font_color2`	color; color of font for high entries if `incl_val` is `TRUE`. `"white"` by default
`font_thres`	numeric; values less than `font_thres` will be displayed in `font_color1` if `incl_val` is `TRUE`. `mean(x)` by default

Value

Returns `ggplot2` visualization of matrix

Examples

```
library(ggplot2)

# get the data
set.seed(12)
data <- generateData()
X <- data$X
Z <- data$Z
interval <- data$interval
prec <- data$true_precision

# get overall and within interval sample sizes
n <- nrow(X)
n1 <- sum(interval == 1)
n2 <- sum(interval == 2)
n3 <- sum(interval == 3)
```

```
# visualize the distribution of the extraneous covariate
ggplot(data.frame(Z = Z, interval = as.factor(interval))) +
  geom_histogram(aes(Z, fill = interval), color = "black", bins = n %/% 5)

# visualize the true precision matrices in each of the intervals

# interval 1
matViz(prec[[1]], incl_val = TRUE) +
  ggtitle(paste0("True precision matrix, interval 1, observations 1,...,", n1))

# interval 2 (varies continuously with Z)
cat("\nInterval 2, observations ", n1 + 1, ",...,", n1 + n2, sep = "")
int2_mats <- prec[interval == 2]
int2_inds <- c(5, n2 %/% 2, n2 - 5)
lapply(int2_inds, function(j) matViz(int2_mats[[j]], incl_val = TRUE) +
         ggtitle(paste("True precision matrix, interval 2, observation", j + n1)))

# interval 3
matViz(prec[[length(prec)]], incl_val = TRUE) +
  ggtitle(paste0("True precision matrix, interval 3, observations ",
                 n1 + n2 + 1, ",...,", n1 + n2 + n3))

# fit the model and visualize the estimated graphs
(out <- covdepGE(X, Z, nssq = 2, nsbsq = 2, npip = 2))
plot(out)

# visualize the posterior inclusion probabilities for variables (1, 3) and (1, 2)
inclusionCurve(out, 1, 2)
inclusionCurve(out, 1, 3)
```

`plot.covdepGE` *Plot the Graphs Estimated by* `covdepGE`

Description

Create a list of the unique graphs estimated by `covdepGE`

Usage

```
## S3 method for class 'covdepGE'
plot(x, graph_colors = NULL, title_sum = TRUE, ...)
```

Arguments

`x`	object of class `covdepGE`; return of `covdepGE` function
`graph_colors`	`NULL` OR vector; the j-th element is the color for the j-th graph. If `NULL`, all graphs will be colored with `"#500000"`. `NULL` by default
`title_sum`	logical; if `TRUE` the indices of the observations corresponding to the graph will be included in the title. `TRUE` by default
`...`	additional arguments will be ignored

Value

Returns list of ggplot2 visualizations of unique graphs estimated by covdepGE

Examples

```
library(ggplot2)

# get the data
set.seed(12)
data <- generateData()
X <- data$X
Z <- data$Z
interval <- data$interval
prec <- data$true_precision

# get overall and within interval sample sizes
n <- nrow(X)
n1 <- sum(interval == 1)
n2 <- sum(interval == 2)
n3 <- sum(interval == 3)

# visualize the distribution of the extraneous covariate
ggplot(data.frame(Z = Z, interval = as.factor(interval))) +
  geom_histogram(aes(Z, fill = interval), color = "black", bins = n %/% 5)

# visualize the true precision matrices in each of the intervals

# interval 1
matViz(prec[[1]], incl_val = TRUE) +
  ggtitle(paste0("True precision matrix, interval 1, observations 1,...,", n1))

# interval 2 (varies continuously with Z)
cat("\nInterval 2, observations ", n1 + 1, ",...,", n1 + n2, sep = "")
int2_mats <- prec[interval == 2]
int2_inds <- c(5, n2 %/% 2, n2 - 5)
lapply(int2_inds, function(j) matViz(int2_mats[[j]], incl_val = TRUE) +
         ggtitle(paste("True precision matrix, interval 2, observation", j + n1)))

# interval 3
matViz(prec[[length(prec)]], incl_val = TRUE) +
  ggtitle(paste0("True precision matrix, interval 3, observations ",
                 n1 + n2 + 1, ",...,", n1 + n2 + n3))

# fit the model and visualize the estimated graphs
(out <- covdepGE(X, Z, nssq = 2, nsbsq = 2, npip = 2))
plot(out)

# visualize the posterior inclusion probabilities for variables (1, 3) and (1, 2)
inclusionCurve(out, 1, 2)
inclusionCurve(out, 1, 3)
```

Index